Software as a Service: Undo

Hernán Merlino
Information Systems Research Group. National University of Lanús, Buenos Aires, Argentine.
hmerlino@gmail.com

Oscar Dieste
Empirical Software Research Group (GrISE). School of Computer Science. Madrid Politechnic University. Spain.
odieste@fi.upm.es

Patricia Pesado
PhD Program on Computer Science. Computer Science School. National University of La Plata. Argentine.
ppesado@lidi.info.unlp.edu.ar

Ramon García-Martínez
Information Systems Research Group. National University of Lanús, Buenos Aires, Argentine.
rgarcia@unla.edu.ar

***Abstract*— This paper proposes a highly automated mechanism to build an undo facility into a new or existing system easily. Our proposal is based on the observation that for a large set of operators it is not necessary to store in-memory object states or executed system commands to undo an action; the storage of input data is instead enough. This strategy simplifies greatly the design of the undo process and encapsulates most of the functionalities required in a framework structure similar to the many object-oriented programming frameworks.**

Keywords- Undo Framework; Software as a Services; and Usability component.

I. INTRODUCTION

It is hard to build usability into a system. One of the main reasons is that this is usually done at an advanced stage of system development [1], when there is little time left and the key designed decisions have already been taken. Usability patterns were conceived with the aim of making usable software development simpler and more predictable [2]. Usability patterns can be defined as mechanisms that could be used during system design to provide the software with a specific usability feature [1]. Some usability patterns defined in the literature are: Feedback, Undo/Cancel, Form/Field Validation, Wizard, User profile and Help [3]. The main stumbling block for applying these patterns is that there are no frameworks or even architectural or designed patterns associated with the usability patterns. This means that the pattern has to be implemented ad hoc in each system. Ultimately, this implies that (1) either the cost of system development will increase as a result of the heavier workload caused by the design and implementation of the usability features or, more likely; (2) many of these usability features (Undo, Wizard, etc.) will be left out in an attempt to reduce the development effort.

The goal of this paper is to develop a framework for one of the above usability patterns, namely, the undo pattern. The undo pattern provides the functionality necessary to undo actions taken by system users. Undo is a common usability features in the literature [4]. This is more than enough justification for dealing with this pattern first. There are other, more technical grounds to support the decision to tackle undo in first place. One of the most important is undoubtedly that undo shares much of its infrastructure (design, code) with other patterns. Redo and cancel are obvious cases, but it also applies to apparently unrelated patterns, like feedback and wizard.

Several authors have proposed alternatives of undo pattern, these alternatives focus on particular applications, notably document editors [5][6] although the underlying concepts are easily exportable to other domains. However, these proposals are defined at high level, without an implementation (or design) reusable in different types of systems. These proposals therefore do not solve the problem of introduction of usability features in software

In this paper, we present a new approach for the implementation of Undo pattern. Our proposal solves a subset of cases (stateless operations) in a highly efficiently manner. The importance of having an automated solution of those is that they are the most frequents operations occur in information systems.

We have implemented the framework using Software as a Service (SaaS). For this class of development we have developed a framework similar to other such as Spring [7] or Hibernante [8] that allows to build the undo easy into a system (that we term “host application”). Furthermore, in host application, it’s only need to include a few modifications in code, and this creates a lower propensity to introduce bugs in the code and allows inclusion of it in a more simple developed system.

This article is structured as follows. Section 2 describes the state of the art regarding the implementation of undo. Section 3 presents the undo infrastructure, whereas Section 4 describes undo infrastructure. Section 5 shows a proof of concept of the proposed framework. Finally, Section 6 briefly discusses and presents the main contributions of our work.

II. BACKGROUND

Undo is a very widespread feature, and is prominent across the whole range of graphical or textual editors, like, for example, word processors, spreadsheets, graphics editors, etc. Not unnaturally a lot of the undo-related work to date has focused on one or other of the above applications. For example, [6] and Baker and Storisteanu [9] have patented two methods for implementing undo in document editors within single-user environments.

There are specific solutions for group text editors that support undo functionality such as in Sun [10] y Chen and Sun [11] and Yang [12]. The most likely reason for the boom of work on undo in the context of document editors is its relative simplicity. Conceptually speaking, an editor is a container

accommodating objects with certain properties (shape, position, etc.). Consequently, undo is relatively easy to implement, as basically it involves storing the state of the container in time units i, i+1, …, i + n. Then when the undo command is received, the container runs in reverse i + n, i + n-1, i.

A derivation of the proposed solutions for text editors is an alternative implementation of undo for email systems like Brown and David [13], these solutions are only for text editors and email systems and applications that are built considering undo functionality from the design.

The problems of undo in multi-user environments have also attracted significant attention. Both Qin [15] and Abrams and Oppenheim [14] have proposed mechanisms for using undo in distributed environments, and Abowd and Dix [4] proposed a formal framework for this field.

In distributed environments, the solution has to deal with the complexity of updates to shared data (basically, a history file of changes) [15].

Several papers have provided insight on the internal aspects of undo, such [16], who attempted to describe the undo process features. Likewise, Berlage [17] proposed the construction of an undo method in command-based graphical environments, Burke [18] created an undo infrastructure, and Korenshtein [19] defined a selective undo.

There has been work done on multi-level models for Undo where each action for a system is defined as a discreet group of commands performed, where each command represents a requested action by the user, this is a really valid approximation because defined as a discreet group of commands, the system could be reverted to any previous stage, only performing the actions the other way round; here a difference can be found between the theory and the practice, regarding the first one it is true that is possible to go back to any previous stage of the system if there is the necessary infrastructure for the Undo, but actually the combination of certain procedures performed by the user or a group of them could be impossible to be solved related to expected response time. For this reason the implementation of the Undo process must complete these possible alternatives with regards to the command combinations performed by the user or users.

Another important aspect which has been worked out is the method of representation of the actions performed by the users in Washizaki and Fukazawa [20], a dynamic structure of commands is presented and it represents the history of commands implemented.

The Undo model representation through graphs has been widely developed in Berlage [17] present a distinction between the linear and nonlinear undo, the nonlinear approach is represented by a tree graph, where you can open different branches according to user actions. Edwards [21] also presented a graph structure where unlike Berlage [17] these branches can be back together as the actions taken. Dix [22] showed a cube-shaped graph to represent history of actions taken. Edwards [23] actions are represented in parallel. It has also used the concept of Milestoning and Rollback [24] to manage the log where actions temporarily stored. Milestoning is a logical process which makes a particular state of the artifacts stored in the log; and rollback is process of returning back the log to one of the points of Milestoning. All these alternative representation of the commands executed by users are valid, but this implementation is not a simple task, because create a new branch and join two existing branches is not a trivial action, because you must know all possible ways that users can take; by this it may be more advisable to generate a linear structure, that can be shared by several users, ordered by time, this structure can be a queue, which is easy to deploy and manage.

Historically frameworks that have been used to represent the Undo only have used the pattern Command Processor [25], Fayard, Shumidt [26] and Meshorer [27]. This serves to keep a list of commands executed by the user, but it is not enough to create a framework that is easy to add to existing systems, As detailed below using service model allows greater flexibility for the undo process integration in an application, this approach allow a greater degree of complexity in the process of allowing Undo handle different configurations.

Undo processes has been associated to exception mechanisms to reverse the function failed [28] these are only invoked before the request fails and the user, these are associated with a particular set of applications.

Patents, like the method for building an undo and redo process into a system, have been registered [29]. Interestingly, this paper presents the opposite of an undo process, namely redo, which does again what the undo previously reverted. Other authors address the complexities of undo/redo as well. Thus, for example, Nakajima and Wash [30] define a mechanism for managing a multi-level undo/redo system, Li [31] describes an undo and redo algorithm and Martinez and Rhan [32] present a method for graphically administering undo and redo, based primarily on the undo method graphical interface.

The biggest problem with the above works is that, again, they are hard to adopt in software development processes outside the document editor domain. The only noteworthy exception to this is a design-level mechanism called Memento [33]. This pattern restores an object to a previous state and provides an implementation-independent mechanism that can be easily integrated into a system. The downside is that this pattern is not easy to build into an existing system. Additionally, Memento only restores an object to a previous state; it does not consider any of the other options that an undo pattern should include.

The solutions presented are optimized for particular cases and are difficult to apply to other domains; on the other hand, it is necessary to include a lot of code associated with Undo in host application.

III. THEORETICAL JUSTIFICATION OF UNDO FRAMEWORK

Before describing proposed Undo Framework, and its implementation as SaaS, theoretical foundations that demonstrate the correctness of our approach. This will be done in two steps; first we will describe how to undo operations that do not depend on its state, the procedure to undo these

operations consist in reinjection input data at time t-1, second we prove that reinjection input always produces correct results.

A. Initial Description

The most commonly used option for developing an undo process is to save the states of objects that are liable to undergo an undo process before they are put through any operation; this is the command that changes the value of any of their attributes. This method has an evident advantage; the system can revert without having to enact a special-purpose process; it is only necessary to remove and replace the current in-memory objects with objects saved previously.

This approach is a simple mechanism for implementing the undo process, although it has some weaknesses. On one hand, saving all the objects generates quite a heavy system workload. On the other hand, developer's need to create explicitly commands for all operations systems. Finally, the system interfaces (mainly the user interface) have to be synchronized with the application objects to enact an undo process. This is by no means easy to do in monolithic systems, but, in modern distributed computer systems, where applications are composed of multiple components all running in parallel (for example, J2EE technology-based EJB), the complications increase exponentially.

There is a second option for implementing an undo process. This is to store the operations performed by the system instead of the changes made to the objects by these operations. In this case, the undo would execute the inverse operations in reverse order. However, this strategy is seldom used for two reasons. On one hand, except for a few exceptions like the above word processing or spreadsheet software, applications are seldom designed as a set of operations. On the other hand, some operations do not have a well-defined inverse (imagine calculating the square of a table cell; the inverse square could be both a positive and a negative number).

The approach that we propose is based on this last strategy, albeit with a simplified complexity. The key is that, in any software system whatsoever, the only commands processed that are relevant to the undo process are the ones that update the model data (for example, a data entry in a field of a form that updates an object attribute, the entry of a backspace character that deletes a letter of a document object, etc.). In most cases, such updates are idempotent, that is, the effects of the entry do not depend on the state history. This applies to the form in the above example (but not, for example, to the word processor). When the updates are idempotent, neither states of the objects in the model nor the executed operations has to be stored, and the list of system inputs is only required. In other words, executing an undo at time t is equivalent to entering via the respective interface (usually the user interface) the data item entered in the system at time t-2. Figure 1 shows an example of this approach. At time t, the user realizes that he has made a mistake updating the name field in the form, which should contain the value John not Sam. As a result, he wants to revert to the value of the field that the form had at time t-1. To do this, it is necessary (and enough) to re-enter the value previously entered at time t-2 in the name field.

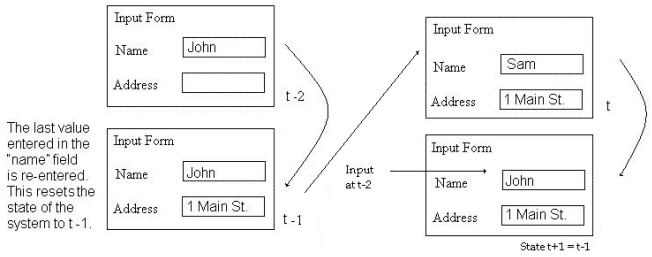

Figure 1. Undo sequence.

Unless the updates are idempotent, this strategy is not valid (as in the case of the word processor, for example), and the original strategy has to be used (that is, store the command and apply its inverse to execute the undo). However, the overwhelming majority of cases executed by a system are idempotent, whereas the others are more of an exception.

Consequently, the approach that we propose has several benefits: (1) the actual data inputs can be processed fully automatically and transparently of the host application; (2) it avoids having to deal with the complexity of in-memory objects; (3) the required knowledge of system logic is confined to commands, and (4), finally, through this approach, it is possible to design an undo framework that is independent of the application and, therefore, highly reusable.

B. Formal Description

The following definitions and propositions are used to proof (in an algebraic way) that UNDO process (UNDO transformation) may be built under certain process (transformation) domain constrains.

Definition 1. Let $E = \{\varepsilon_j^i / \varepsilon_j$ is a data structure$\}$ be the set of all data structures.

Definition 2. Let ε_j^i be the instance i of data structure ε_j belonging to E.

Definition 3. Let $\varepsilon_j^C = \{ \varepsilon_j^i / \varepsilon_j^i$ is an instance i of the structure $\varepsilon_j\}$ be the set of all the possible instances of data structure ε_j.

Definition 4. Let $o_\tau^{\varepsilon j}$ be a transformation which verifies $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^C$ and $o_\tau^{\varepsilon j}(\varepsilon_j^i) = \varepsilon_j^{i+1}$.

Definition 5. Let ε_j^{Cr} be a constrain of ε_j^C defined as $\varepsilon_j^{Cr} = \{ \varepsilon_j^i / \varepsilon_j^i$ is an instance i of the data structure ε_j which verifies $o_\tau^{\varepsilon j}(\varepsilon_j^{i-1}) = \varepsilon_j^i\}$

Proposition 1. If $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$ then $o_\tau^{\varepsilon j}$ is bijective. Proof: $o_\tau^{\varepsilon j}$ es injective by definition 4, $o_\tau^{\varepsilon j}$ is surjective by definition 5, then $o_\tau^{\varepsilon j}$ is bijective for being injective and surjective. QED.

Proposition 2. If $o_\tau^{\varepsilon j} : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$ then has inverse. Proof: Let $o_\tau^{\varepsilon j}$ be bijective by proposition 1, then by usual algebraic properties $o_\tau^{\varepsilon j}$ has inverse. QED.

Definition 6. Let o_τ be the set of al transformations $o_\tau^{\varepsilon j}$.

Definition 7. Let Φ be the operation of composition defined as usual composition of algebraic transformations.

Definition 8. Let Σ be the service defined by structure < E^{Σ} , o_{τ}^{Σ} , Φ > where $E^{\Sigma} \subseteq E$ and $o_{\tau}^{\Sigma} \subseteq o_{\tau}$.

Definition 9. Let $X = o_{\tau}^{\varepsilon j1} \Phi o_{\tau}^{\varepsilon j2} \Phi \ldots \Phi o_{\tau}^{\varepsilon jn}$ be a composition of transformations which verifies $o_{\tau}^{\varepsilon ji} : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$ for all i:1...n. By algebraic construction $X : \varepsilon_j^C \rightarrow \varepsilon_j^{Cr}$.

Proposition 3. The composition of transformations X has inverse and is bijective. Proof: Let be $X = o_{\tau}^{\varepsilon j1} \Phi o_{\tau}^{\varepsilon j2} \Phi \ldots \Phi o_{\tau}^{\varepsilon jn}$. For all i:1...n verifies $o_{\tau}^{\varepsilon ji}$ has inverse by proposition 2. Let $[o_{\tau}^{\varepsilon ji}]^{-1}$ be the inverse transformation of $o_{\tau}^{\varepsilon ji}$, by usual algebraic properties $[o_{\tau}^{\varepsilon ji}]^{-1}$ is bijective. Then it is possible to compose a transformation $X^{-1} = [o_{\tau}^{\varepsilon jn}]^{-1} \Phi [o_{\tau}^{\varepsilon jn-1}]^{-1} \Phi \ldots \Phi [o_{\tau}^{\varepsilon j1}]^{-1}$. The transformation X^{-1} is bijective by being composition of bijective transformations. Then transformation $X^{-1} : \varepsilon_j^{Cr} \rightarrow \varepsilon_j^C$ exists and is the inverse of X. QED.

Definition 10. Let UNDO be the X^{-1} transformation of X.

IV. Structure of Undo Framework

In this section, we will describe our proposal for designing the undo pattern using SaaS to implement the replay of data.

A. Undo Service Architecture

Figure 2 represents the service Undo infrastructure, a high-level abstraction of the architecture. Undo service has 3 modules, (a) Undo Business Layer, (b) Undo Application Layer (c) Undo Technology Layer.

Undo Business Layer is responsible for creating, maintaining and deleting applications that will access the undo service. An application that could access to service must execute following steps: (a) creating application unique identifier, this should be attached to each message that is sent to the service, (b) creation of user profile identifier, this must be attached to each message that is sent to the service, once defined two identifiers, host application may immediately use undo service

All these added to the header data set that can be invoked by the user for later retrieval, enable the service to handle different applications at the same time, within an application users can manage their own recovery without interfering lists, plus each user can manage their own separate lists per interface; the service giving maximum flexibility for every application.

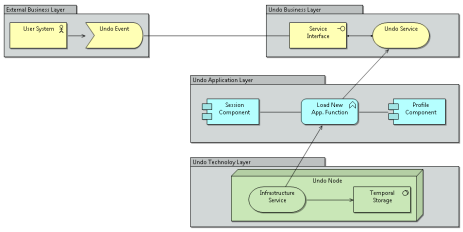

Figure 2. Undo infrastructure

B. Operation of Undo Framework

Fig. 3 details process of send and get data to service, first we described undo receives data service from external system, at this point is where you start the process that ended with the injection of data re be invoked by the external system. In the External Layer, the user application generates an event that triggers an action likely to be overturned, this creates an Undo Service invocation, this is received by the service interface that is plotted on the Undo Abstract Layer, this action fires a set of processes:

(i) Check current user session, this start with Validate Session and Profile, this process communicates with the Undo Application Layer, with function that processes Validate Undo Service Session and Profile. This service is based on two components responsible for validation and maintenance of active user sessions and profile´s user, Session component is responsible for validating whether the session with which you access the service is active, component Profile is responsible for validating invocation of the temporary storage. Both components communicate with lower-level layer called Layer Undo Technology, this is basic infrastructure for Undo service, which consists of a processing unit and data storage.

(ii) After that, the validation process begins to check if host application has access to temporary storage, this process communicates with Validate Undo Data process, and it is responsible for validating the data to be stored, first validates that host application is active, if so, host application obtains credentials to use. If the process is successful the user is returned a successful update code, if an error occurs, it returns an error code also asynchronously, and with external system code decides if it generates exception or continues with the normal flow.

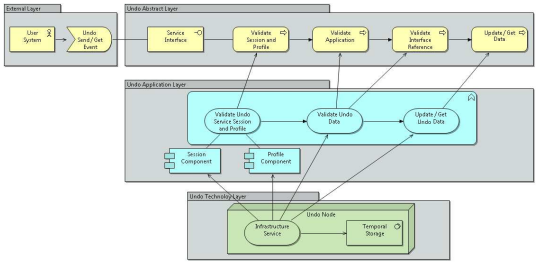

Figure 3. Undo receives data service

At last we described process which undo data service return stored temporarily to external system, this is where it describes the beginning of re injection data by the external system for the service provided. In the same way explained above, Layer External triggers an event that generates a request for data stored, this process is divided into two stages:

(i) Charge of the validation of the application, which has the same activities as described in the process of Undo Send Data,

(ii) Retrieve all values that have been stored for the tuple, application and interface. Return of this process to External Layer is the list sorted in reverse with all values stored, service provide option to request only the last value

stored. If event failure, external application receives an asynchronously error code.

V. CONCLUSIONS

In this paper we have proposed the design of an undo framework to build the undo functionality into any software application whatsoever. The most salient feature of this framework is the type of information it stores to be able to undo the user operations: input data instead of in-memory object states or commands executed by the system. This lessens the impact of building the framework into the target application a great deal.

Building an Undo Service has some significant advantages with respect to Undo models presented, first of all the simplicity of inclusion in a host application under construction or existing, you can see in the proof of concept. Second the independence of service in relation to the host application allows the same architectural model to provide answers to different applications in different domains. Construction of a service allows to Undo be a complex application, with possibility of include analysis for process improvement, as described in the next paragraph it is possible to detect patterns of invocation of Undo in different applications.

Further work is going to bring: (a) creation of a pre-compiler, (8) automatic detection of fields to store, (c) extend the framework to other platforms.

VI. REFERENCES

1. Ferre, X., Juristo, N., Moreno, A., Sanchez, I. 2003. A Software Architectural View of Usability Patterns. 2nd Workshop on Software and Usability Cross-Pollination (at INTERACT'03) Zurich (Switzerland)
2. Ferre, X; Juristo, N; and Moreno, A. 2004. Framework for Integrating Usability Practices into the Software Process. Madrid Polit. University.
3. Juristo, N; Moreno, A; Sanchez-Segura, M; Davis, A. 2005. Gathering Usability Information through Elicitation Patterns.
4. Abowd, G.; Dix, A. 1991. Giving UNDO attention. University of York.
5. Qin, X. y Sun, C. 2001. Efficient Recovery algorithm in Real-Time and Fault-Tolerant Collaborative Editing Systems. School of computing and Information Technology Griffith University Australia.
6. Bates, C. and Ryan, M. 2000. Method and system for UNDOing edits with selected portion of electronic documents. PN: 6.108.668 US.
7. Spring framework. http://www.springsource.org/.
8. Hibernate framework. http://www.hibernate.org/.
9. Baker, B. and Storisteanu, A. 2001. Text edit system with enhanced UNDO user interface. PN: 6.185.591 US.
10. Sun, C. 2000. Undo any operation at time in group editors. School of Computing and Information Technology, Griffith University Australia.
11. Chen, D; Sun, C. 2001. Undoing Any Operation in Collaborative Graphics Editing Systems. School of Computing and Information Technology, Griffith University Australia.
12. Yang, J; Gu, N; Wu, X. 2004. A Documento mark Based Method Supporting Group Undo. Department of Computing and Information Technology. Fudan University.
13. Brown, A; Patterson, D, 2003. Undo for Operators: Building an Undoable E-mail Store. University of California, Berkeley. EECS Computer Science Division.
14. Abrams, S. and Oppenheim, D. 2001. Method and apparatus for combining UNDO and redo contexts in a distributed access environment. PN: 6.192.378 US.
15. Berlage, T; Genau, A. 1993. From Undo to Multi-User Applications. German National Research Center for Computer Science.
16. Mancini, R., Dix, A., Levialdi, S. 1996. Reflections on UNDO. University of Rome.
17. Berlage. T. 1994. A selective UNDO Mechanism for Graphical User Interfaces Based On command Objects. German National Research Center for Computer Sc.
18. Burke, S. 2007. UNDO infrastructure. PN: 7.207.034 US.
19. Korenshtein, R. 2003. Selective UNDO. PN: 6.523.134 US.
20. Washizaki, H; Fukazawa, Y. 2002. Dynamic Hierarchical Undo Facility in a Fine-Grained Component Environment. Department of InformaTION AND Computer Science, Waswda University. Japan.
21. Edwards, W; Mynatt, E. 1998. Timewarp: Techniques for Autonomous Collaboration. Xerox Palo Alto Research Center.
22. Dix, A; Mancini, R; Levialdi, S. 1997. The cube – extending systems for undo. School of Computing, Staffordshire University. UK.
23. Edwards, W: Igarashi, T; La Marca, Anthony; Mynatt, E. 2000. A Temporal Model for Multi-Level Undo and Redo.
24. O´Brain, J; Shapiro, M. 2004. Undo for anyone, anywhere, anytime. Microsoft Res..
25. Buschmann, F; Meunier, R; Rohnert, H; Sommerlad, P; Stal, M. 1996. Pattern-Oriented Software Architecture: A System Of Patterns. John Wiley & Sons.
26. Fayad, M.; Shumidt, D. 1997. Object Oriented Application Frameworks. Comunications of the ACM, 40(10) pp 32-38.
27. Meshorer, T. 1998. Add an undo/redo function to you Java app with Swing. JavaWord, June, IDG Communications.
28. Shinnar, A; Tarditi, D; Plesko, M; Steensgaard, B. 2004. Integrating support for undo with exception handling. Microsoft Research.
29. Keane, P. and Mitchell, K. 1996. Method of and system for providing application programs with an UNDO/redo function. PN:5.481.710 US.
30. Nakajima, S., Wash, B. 1997. Multiple level UNDO/redo mechanism. PN: 5.659.747 US.
31. Li, C. 2006. UNDO/redo algorithm for a computer program. PN: 7.003.695 US.
32. Martinez, A. and Rhan, M. 2000. Figureical UNDO/redo manager and method. PN: 6.111.575 US.
33. Gamma, E., R. Helm, R. Johnson, and J. Vlissides. 1994. Design Patterns: Elements of Reusable Object-Oriented Software, Addison- Wesley.